1. Уровни языков программирования.
2. Структурное программирование. Принципы структурного программирования.
3. Язык С/С++. Общие сведения и характеристика языка, состав языка
4. Алгоритм. Способы записи алгоритма (блок-схемы, псевдокод, ...)
5. Структура программы на языке С/С++. Пример простой программы.
6. Принципы типизации данных. Иерархия простых типов данных.
7. Стандартные типы данных. Таблица характеристик. Особенности выбора типа.
8. Внутреннее представление данных типа int
9. Внутреннее представление данных с плавающей точкой
10. Явное и неявное преобразование типов. Правила преобразования типов.
11. Переменные (объявление, инициализация, присвоение).
12. Константы. Специальные символы. Квалификатор const.
13. Типы данных, определяемые пользователем. Перечисляемый тип (enum)
14. Структуры (назначение, синтаксис, использование, оператор ->).
15. Область видимости переменных.
16. Операции (классификация, особенности записи, таблица приоритетов). Понятие ассоциативности, приоритета, размерности. Операция sizeof.
17. Арифметические операции
18. Логические операции и операции сравнения
19. Поразрядные операции
20. Операция присваивания (синтаксис, логика работы, полная и сокращённая форма, порядок выполнения, контекст вычисления, пример).
21. Операция ?: (синтаксис, логика работы, пример). Отличие от условного оператора
22. Операторы языка С. Пустой оператор, составной оператор
23. Условный оператор (полная и краткая формы, неоднозначность условного оператора). Связь между несколькими условными операторами и сложными логическими выражениями.
24. Оператор выбора (синтаксис, логика работы, пример). Оператор break
25. Цикл с предусловием. Цикл с постусловием.
26. Цикл с параметром. Взаимозаменяемость циклов.
27. Операторы передачи управления goto, break и continue
28. Указатели
29. Ссылки. Различие между указателями и ссылками
30. Одномерные массивы (объявление, индексация, хранение в памяти). Типовые алгоритмы обработки массивов.
31. Двумерные массивы (статические и динамические)
32. Массивы и их связь с указателями. Адресная арифметика
33. Передача массивов в качестве параметров функции
34. Концепция памяти. Операции для работы с динамической памятью. Выделение и освобождение памяти под переменные, одномерные массивы.
35. Подпрограммы (синтаксис, виды подпрограмм, контекст, пример).
36. Объявление и определение функций. Оператор return.
37. Способы передачи параметров в функцию
38. Способы передачи значения из одной функции в другую
39. Параметры функции со значениями по умолчанию
40. Функции с переменным числом параметров
41. Перегрузка функций
42. Функция main(). Передача параметров в функцию main.
43. Указатель на функцию. Передача имен функций в качестве параметров
44. Представление строк в языках программирования. Строки в С. Основные алгоритмы обработки строк.
45. Работа с символами. Основные функции стандартной библиотеки <cctype>.
46. Работа со строками. Основные функции стандартной библиотеки <cstring>.
47. Файловый ввод-вывод. Стандартная библиотека ввода-вывода (cstdio). Типовые алгоритмы обработки файлов.

48. Потоковый ввод-вывод. Файловые потоки (fstream). Типовые алгоритмы обработки файлов.
49. Потоковый ввод-вывод. Основные функции управления вводом-выводом библиотеки <iostream>.
50. Потоковый ввод-вывод. Форматирование данных (функции форматирования и манипуляторы).
51. Основные функции управления вводом-выводом <cstdio>.



1. Уровни языков программирования.

Низкий уровень – языки ориентированные на конкретный тип процессора (операторы языка близки к машин.коду и ориентированы на конкретные команды процессора). Язык самого низкого уровня – ассемблер. Применяются, в основном, для написания операционных систем, драйверов.

Высокий уровень – языки более понятны человеку. Особенности конкретных компьютерных архитектур не учитываются, потому лёгкая переносимость. Проще разработка и меньшее кол-во ошибок.

Транслятор – программа для перевода алгоритма решения на языке высокого уровня в машинный код. (Если трансляция сопровожд. выполнением каждой команды после её перевода в маш.код – это интерпретатор, если только переводом программы в маш.код – компилятор).

2. Структурное программирование. Принципы структурного программирования.

Это методология разработки программного обеспечения, в основе которой лежит представление программы в виде иерархической структуры блоков. Предложена в 1970-х Э. Дейкстрой.

Принцип 1. Отказ от использования оператора безусловного перехода goto.

Принцип 2. Любая программа строится из 3 базовых управл. конструкций: последовательность (однократное выполнение операций в том порядке, в котором они записаны в тексте программы), ветвление (однократное выполнение операций в том порядке, в котором они записаны в тексте программы) и цикл (многократное исполнение одной и той же операции, пока выполняется задан. Условие).

Принцип 3. Базовые управл. конструкции могут быть вложены друг в друга произвольным образом.

Принцип 4. Повторяющиеся фрагменты программ оформляются в виде подпрограмм (ф-ций и процедур).

Принцип 5. Каждая логически законченная группа инструкций оформляется как блок (блоки – основа структ. программирования)

Принцип 6. Все перечисленные конструкции должны иметь один вход и один выход.

Принцип 7. Разработка программы ведётся пошагово, методом «сверху вниз».

3. Язык С/С++. Общие сведения и характеристика языка, состав языка

Язык C/C++ – высокоуровневый компилируемый статически типизированный ЯП общего назначения. Поддерживает такие парадигмы программирования как процедурное программирование, объектно-ориентированное *(C++)* и обобщённое программирование. Автор языка C++ – Бьёрн Страуструп. Языка Си – Деннис Ритчи.

4. Алгоритм. Способы записи алгоритма (блок-схемы, псевдокод, ..)

Алгоритм – точная инструкция исполнителю в понятной для него форме, определяющая процесс достижения поставленной цели на основе имеющихся исходных данных за конечное число шагов.

Способы записи алгоритмов: вербальный – когда он описывается на человеческом языке, символьный – с помощью набора символов и графический – с помощью набора граф. изображений.

Описание алгоритма с помощью блок-схем осуществляется рисованием последовательности геометрических фигур, каждая из которых подразумевает выполнение определенного действия алгоритма. Порядок выполнения задается стрелками. Регламентируется ГОСТом.

В зависимости от последовательности выполнения действий выделяют алгоритмы линейной, разветвленной и циклической структуры. Линейная структура (действия выполняются послед. одно за другим):

Разветвлённая структура (выполнение действий в зависимости от условия; каждая такая послед. – ветвь алгоритма):

Циклическая структура (выполнение повторяющейся последов. действий в зависимости от условия; такие действия – тело цикла). Различают циклы с предусловием и постусловием:

Псевдокод – компатный язык описания алгоритмов, использующий ключ. Слова ЯП, но опускающий несущ. подробности и специфический синтаксис.

5. Структура программы на языке C/C++. Пример простой программы.

Программа на языке C/C++ состоит из функций, описаний и директив препроцессора. Одна из ф-ций должна иметь имя main. Выполнение программы начинается с первого оператора этой ф-ции.

Лексемы: идентификаторы, ключевые слова, константы, строки, операторы и прочие разделители.

Разделители: символы пробела, табуляции, новой строки.

Комментарии: /* многострчные */, //одностр

Операторы: выражения – любое выраж., после которого стоит «;» считается компилятором как отдельный оператор, пустой оператор «;», составной оператор или блок – «{список операторов}»

Строки: последовательности символов, заключенные в двойные кавычки. Последний символ – \0

Ключевые слова: слова, зарезервированные для спец. Целей, которые не должны быть использ. В качестве обычных имен идентификаторов.

Идентификаторы: имена переменных, ф-ций.

Пример программы выводящей Hello, World:

```
#include <iostream>
int main (){std::cout << "Hello, World!\n";}
```

6. Принципы типизации данных. Иерархия простых типов данных.

Тип данных определяет внутреннее представление данных в памяти компьютера, а также определяет мн-во допустимых значений. Типы данных бывают простые и составные. Простые: целые, вещественные, символьные, логические, перечисляемые, указатели. Составные: массивы и структуры.

Тип данных **void**: Неопределенный тип. Объектов типа void не существует. Используется для объявления функций не возвращающих значение, указания пустого списка аргументов ф-ции и создания универсального указателя.

Тип данных **bool (C++):** Целочисленный тип данных (логический тип данных). В памяти занимает 1 байт. Целые числа [0;255]. 0 соответствует false, [1;255] – true;

Тип данных **char**: Целочисленный тип данных, используется для представления символов. В памяти занимает 1 байт.

Тип данных **int**: Целочисленный тип данных. Занимает в памяти 4 байта.

Тип данных **float**: Тип данных с плавающей точкой. Занимает в памяти 4 байта.

Тип данных **double**: Тип данных с плавающей точкой. Занимает в памяти 8 байт.

Возможны вариации типа **int** с приставками-спецификаторами. Спецификаторы знаковости (для целочисленных типов): **unsigned – беззнаковое, signed – знаковое (увеличивает в два раза количество положительных значений).** Спецификаторы размера: **short – целевой тип будет не менее 2 байт, long – не менее 4 байт, long long – не менее 8 байт *(стандарт C++11)*.**

Возможно использование спецификатора long с типом **double.**

Тип		Диапазон	Знач	Эквив.
char	1	-2^7..2^7 - 1	-128.127	Signed char
Unsigned char	1	0..2^8 - 1	0..255	
int	2	-2^15 .. 2^15 - 1		signed
	4	-2^31 .. 2^31 - 1		Signed int
Unsigned int	2	0..2^16 - 1		
	4	0..2^32 - 1		
Short int	2	-2^15 .. 2^15 - 1		Short, signed short int
Unsined short int	2	0..2^16 - 1		
Long int	4	-2^31..2^31 - 1		
Unsigned long int	4	0..2^32 - 1		
Float	4	± (3.4*10^-38 .. 3.4*10^38)	(7-8 знач. Цифр после запятой)	
double	8	± (1.7*10^-308..1.7*10^308)	(15-16 знач. Цифр после запятой)	

7. Стандартные типы данных. Таблица характеристик. Особенности выбора типа.

Смотри вопрос 6.

8. Внутреннее представление данных типа int.

Для представления целых чисел существуют: беззнаковое и со знаком;

В K-разрядной ячейке может храниться 2^K различных значений целых чисел.

Положительное число хранится в прямом коде (двоичное число с дополнением слева незначащими нулями до K разрядов)

Отрицательное число хранится в дополнительном коде: прямой код инвертируется (0 меняются на 1 и обратно) – обратный код, к полученному числу прибавляется 1.

Старший, K-й разряд во внутреннем представлении любого полож. числа = 0, отрицательного = 1. Этот разряд называется знаковым разрядом.

Short int (2 байта):

S (знак)	Значение числа
15	14 0

Unsigned short:

Значение числа
15 0

Long int (int) (4 байта):

S (знак)	Значение числа
31	30 0

Unsigned long int:

Значение числа
31 0

9. Внутреннее представление данных с плавающей точкой.

Данные представляются в виде 2 частей: мантиссы M и порядка P числа в двоичной системе счисления. C = M*2^P



Вещественное число хранится с нормализованной мантиссой, т. е. У которой старший бит всегда равен = 1 (и потому он не хранится в памяти, это неявная единица). Отбрасывание старшей цифры мантиссы выполняется для float и double, но не для long double.

Порядок числа хранится «сдвинутым», т. е. К нему прибавляется число так, чтобы порядок был всегда неотрицательный. Для чисел формата float прибавляется 127, для double – 1023. Число **float**:

S (знак)	P (8 бит)	M (23 бита)
31	30 23	22 0

Число **double**:

S (знак)	P (11 бит)	M (52 бита)
63	62 52	51 0

10. Явное и неявное преобразование типов. Правила преобразования типов.

Неявное преобразование выполняет транслятор, явное – сам программист.

Результат любого вычисления будет преобразовываться к наиболее точному типу данных, из тех, которые участвуют в вычислении.

В С++ предусмотрена унарная операция приведения типа: *static_cast<тип_данных>(переменная_или_число) (а ещё dynamic_cast, reinterpret_cast)*

Возможно приведение в стиле С:

(тип_данных)переменная_или_число

Или *тип_данных(переменная_или_число)*

Неявное преобразование типом может проходить:

1) При выполнении операции присваивания рез-т приводится к типу вычисления слева от знака (int N=11.1 // N=11)

2) При передаче параметров и возврате знач. Ф-ции они преобразуются в тип, который известен благодаря описанию ф-ции

3) В арифмет. Выраж. Тип результат выражения определяется самым широким типом средив всех операндов (результирующий тип)

11. Переменные (объявление, инициализация, присвоение)

Объявление: сначала указывается тип данных для этой переменной, а затем название этой переменной. Имя переменной может содержать от одного до 32 символов (строчные и прописные буквы, цифры и символ подчеркивания), первым символом обязательно должна быть буква. Имя не должно совпадать с зарезервированными словами. Если переменная объявлена внутри ф-ции – локальная переменная – не видима за пределами своего блока («{}»). Вне ф-ции – глобальная – сохраняет свое значение на протяжении всей программы. Если переменные объявлены как аргументы ф-ций – это формальные параметры ф-ции.

Инициализация: при объявлении переменной можно её проинициализировать, т. е. присвоить нач. значение. Неинициализированные локальные переменные до первого присвоения имеют произвольное значение.

Присваивание: для присваивания служит знак «=». Выражение, стоящее справа от знака присваивания, вычисляется, и полученное значение присваивается переменной стоящей слева от знака присваивания. Предыдущее значение переменной стирается и заменяется на новое. Кроме простого оператора присваивания существуют комбинированные : «+=», «-=», «*=», «/=», «%=». Если необходимо изменить значение переменной на 1, используют инкремент и декремент. Инкремент – увеличивает значение на 1, декремент – уменьшает на 1.

Используют постфиксную (x++ или x--) или префиксную (++x или --x) запись. При использовании *постинкремента* или *постдекремента*, значение переменной сначала используется в выражении, а потом уже изменяется. При использовании *преинкремента* или *предекремента*, значение переменной сначала изменяется, а потом уже используется в выражении.

12. Константы. Специальные символы. Квалификатор const.

Переменная любого типа может быть быть объявлена как немодифицируемая. Это достигается добавлением ключевого слова const к спецификатору типа. Если после const отсутствует спецификатор, то константы рассматриваются как величины со знаком, и им присваивается тип int или long int в соответствии с размером значения.

1. Целые: десятичные, восьмеричные (0, 02 0..7), шестнадцетиричные (0x, 0X 0XFF), двоичные (С++14) (0b, 0B). *(U или u-unsigned, L или l-long, LU; – после числа)*

2. Числа с плавающей точкой: e E (6.21e-5), f F (3.14159F)

3. символьные: „A“, „a“, „5“, „\n“

4. строковые константы

В стиле С: #define имя_конст знач_конст

13. Типы данных, определяемые пользователем. Перечисляемый тип (enum)

Новые имена типов данных можно определять с помощью *typedef опред_типа имя_типа;* (typedef unsigned int uint;)

Перечисления: enum [имя типа] {список значений}:

enum {A, B, C}; (мн-во именованных констант по умолчанию имеют зн-ия 0, 1, 2..)

enum {A=10, B=20, C=30};

enum keys {A, B, C, D}; keys k1, k2; k1 = B;

// k1=1, так как B=1



14. Структуры (назначение, синтаксис, использование, оператор ->)

Структура – совокупность эл-тов произвольных типов, (в отличии от массивов, которые поддерживают элементы одного типа).

Struct [имя_типа] {

тип_1 элемент_1;

тип_2 элемент_2;

…

тип_n элемент_n;

} [список описателей];

Эл-ты структуры – поля структуры, могут иметь любой тип, кроме типа этой же структуры (но могут быть указателями на него). Если отсутствует имя типа, должен быть указан список описателей переменных, указателей или массивов. В этом случае описание структуры служит определением эл-тов этого списка.

Определение массива структур и указ. На структуру:

*Struct { char fio[30]; int date, code; double salary; } staff[100], *ps;*

Ещё пример:

```
struct address {
        char* name;         // имя
        long  number;       // номер дома
        char* street;       // улица
        char* town;         // город
};
```

Для инициализации структуры значения её эл-тов перечисл в фигурных скобках в порядке их описания (address *addr1={«IVANOV», 174, «Nevsky pr.», «Spb»};)*

Доступ к полям структуры выполняется с помощью операций выбора «.» при обращении к полю через имя структуры и

«→» при обращении через указатель.

*address m[10]; address * pa=new address;*

addr1.name="Petrov"; m[0].name=" Kibzun";

pa→number=123;

Для переменных одного и того же структурного типа определена операция присваивания, при которой происходит поэлементное копирование *(address addr2; addr2=addr1;).* Два структурных типа разлины даже когда имеют одни и те же члены.

15. Область видимости переменных.

Если переменная объявлена внутри ф-ции – локальная переменная – не видима за пределами своего блока («{}»). Вне ф-ции – глобальная – сохраняет свое значение на протяжении всей программы.

Если переменные объявлены как аргументы ф-ций – это формальные параметры ф-ции, используются только в ф-ции в которой заданы.

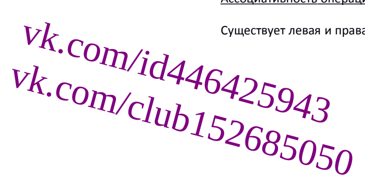
16. Операции (классификация, особенности записи, таблица приоритетов). Понятие ассоциативности, приоритета, размерности. Операция sizeof.

Арность операции – количество операндов, над которыми эта операция выполняется.

Унарная – 1 операнд, бинарная – 2, тернарная – 3 операнда.

Ассоциативность операций – последовательное их выполнения или направление вычислений, при одинаковом приоритете и отсутствии явного указания на очередность их выполнения.

Существует левая и правая ассоциативность.



4 класса операций:

1) арифметические; 2) логические

3) сравнения; 4) присваивания;

Унарные операция (один операнд) и операции присваивания – правая ассоциативность *(порядок выполнения справа-налево)*, все остальные – левая.

Арифметические операции: (+ - * / % ++ - - и унарный минус (меняет знак числа, например: *int x=3; -x; // -3*))

Префиксная и постфиксная форма записи инкремента/декремента: при использовании *постинкремента* или *постдекремента*, значение переменной сначала используется в выражении, а потом уже изменяется. При использовании *преинкремента* или *предекремента*, значение переменной сначала изменяется, а потом уже используется в выражении.

Операции сравнения: (> >= < <= == !=) возвращают 1 если условие истинно, 0 если ложно.

Логические операции: (& - лог. И; || – лог. ИЛИ; ! – лог. ОТРИЦАНИЕ (НЕ);

Приоритет логических операций и операций сравнения: !, (> >= < <=), (== !=), &&, ||

Поразрядные операции: (& – побитовая И; | – побитовая ИЛИ; ^ - поразр. ИСКЛ. ИЛИ;

~ – поразр. ИНВЕРТИРОВАНИЕ; « – побит. Сдвиг влево; » – побит. Сдвиг вправо.

Операции присваивания: (=, специальные форма операции присваивания: +=, -=, *=, /=, %=, >>=, <<=)

Операция sizeof – вычисление размера объектов или типов в байтах.

Условная (тернарная) операция выр. 1 ? выр. 2 : выр.3 (например: *x=1; y=x>9?100:200; //y=100*). Тернарную условную операцию можно применять там, где нельзя использовать оператор if: например при возвращении значения из ф-ции (*например return 5 < 10 ? 1 : 0;)*

17. Арифметические операции

18. Логические операции и операции сравнения

19. Поразрядные операции.

20. Операция присваивания (синтаксис, логика работы, полная и сокращённая форма, порядок выполнения, контекст вычисления, пример)

21. Операция ?: (синтаксис, логика работы, пример). Отличие от условного оператора.

Смотреть 16 билет.

22. Операторы языка С. Пустой оператор, составной оператор.

Пустой оператор – оператор состоящий только из точки с запятой. Не меняет состояние программы.

Составной оператор – {…} – действие составного оператора состоит в послед. Выполнении содержащихся в нем операторов.

Операторы ветвления: if … else и switch

Операторы цикла: с предусловием, с постусловием, с параметром;

Оператор разрыва: break;

Оператор продолжения: continue;

Оператор возврата: return;

23. Условный оператор (полная и краткая формы, неоднозначность условного оператора между несколькими условными операторами и сложными логическими выражениями).

Условный оператор *if (выраж.) оператор1 [else if (выраж.) оператор2] [else оператор3]*

Вход в if должен иметь скалярный результат (целое, с плав. Зап., но не может быть массивом)

24. Оператор выбора (синтаксис, логика работы, пример). Оператор break;

Оператор выбора *switch (выражение) { case конст_выр1: оператор1; case конст_выр2: оператор2; … case конст_вырN: операторN; [default: оператор;]}*

Осуществляет передачу управления на один из нескольких операторов. В операторе switch нет двух одинаковых const.

Оператор break осуществляет остановку выполнения и контроль передачи в оператор после оператора switch. Без него будет выполнен каждый оператор из сопоставленной метки case в конце switch, включая default.

25. Цикл с предусловием. Цикл с постусловием.

С предусловием:

While (выражение) тело_цикла;

выражение – арифм. Тип или приводящий к нему оператор.

Проверяется истинность выражения (условие продолжения цикла) в начале каждой итерации.

С постусловием:

do оператор while(выражение)

Проверяет истинность выражения (условие продолжения цикла) в конце каждой итерации.

26. Цикл с параметром. Взаимозаменяемость циклов.

Цикл с параметром for (оператор_инициализации; провер_выр; обновл_выр) оператор;

Опер. Инициализации исп. Для объявления и присвоения нач. значений величин, использ. В цикле.

Может быть использована операция «,»: j=1, i=1;

Провер_выр. Определеяет усл. Продолжения цикла, вычисляется перед вып. Каждой итерации.

Обновл_выр. Выполняется после каждой итерации для изменения параметров цикла.

Оператор – тело цикла. (пустой, 1 оператор, блок операторов {})

27. Операторы передачи управления goto, break и continue.

Оператор безусловного перехода goto:

goto метка; метка: оператор;

Оператор выхода из цикла break;

Оператор перехода к след. Итерации continue;

28. Указатели.

Тип * имя указателя (int * pint)

void * говорит об отсутствии данных в памяти. Можно присвоить значение указателя любого типа, а также сравнивать его со знач. Любого указателя, перед void * необходимо провести операцию явного приведения типа.

Операции для работы с указателями (адресные операции):

1) & операция взятия адреса. & выражение *(int x=3, *p=&x;)*

2) * операция разыменования указателя. * указатель. Возвращает значение переменной расположенной по указ. Адресу.

Число байт, извлекаемой из памяти опред. Исходя из типа указателя.

Инициализация указателя:

1) присвоение указателю адреса переменной (*int x=3, *p=&x;)*

2) присвоение указателю адреса области памяти в явном виде: *(int * p = 0x0052f93b;)*

Операции для указателей:

1) присваивания 2) арифметические (сложение, вычитание с константой, инкремент, декремент, разность)

3) приведение типов; 3) сравнение;

Типы указателей:

1) указатель на конст. данные – не позволяет модиф. данные на которые они ссылаются

*const тип * имя_ук; или тип const * имя_ук;*

2) константный указатель на неконстантные данные – имеет пост. адрес памяти. *Тип * const имя_указ = адрес;*

3) константный указазатель на константные данные – не позволяет изменять данные на кот. указывает и не позвол. менять адрес.

*const тип * const имя_ук = адрес; или тип const * const имя_ук = адрес;*

29. Ссылки. Различие между указателями и ссылками.

Ссылочные переменные являются псевдонимм для друг. переменной.

Тип & имя_ссылки = имя_переменной;

(int var=100, &r = var; r++;// var = 101, r=101)

Ссылочные переменные должны инициализироваться при объявлении, кроме случаев, когда они является параметром ф-ции.

Тип ссылки должен совпадать с типом переменной, на которую они ссылаются. Ссылки не могут иметь значение NULL (а указатели могут).

Ссылки не могут быть переопределены (а указатели могут).

Ссылки не могут быть неинициализированными (а указатели могут).

Ссылка – это адрес объекта. Указатель – указатель на этот адрес.

30. Одномерные массивы (объявление, индексация, хранение в памяти). Типовые алгоритмы обработки массивов.

Объявление: Тип имя_массива [размерность], где размерность – константа (статич. массив)

const int =10; int m[10];

Имя массива является указателем на его первый элемент. (&m[0] == m[0])

Инициализация эл-тов массива:

1) char c[] = {„A“,“B“};

sizeof (c); // = sizeof(char)*2

2) int b[6] = {1, 2, 3}; // 123000

3) int mos[10] = {} // иниц. массива нулями

Индексация эл-тов массива производится с 0

31. Двумерные массивы (статические и динамические).

Многомерные массивы задаются указанием каждого эл-та в [] скобках.

Тип имя_масс [кол_во строк][кол_во столбцов];

Динамический двумерный массив создается с помощью операций для работы с динамической памятью:

```
int ncol = 10, nrow=10;
double ** mas = new double [nrow]; // выдел. память под массив указателей на строки
for (int i = 0; i < nrow; ++i)
  mas[i] = new int [ncol]; // выдел. память под каждый эл-т массива
for(int i=0; i < nrow; ++i)
   for(int j=0; j < ncol; ++j)
       cin » mas[i][j];
```

for(int i=0; i < nrow; ++i) delete [] mas[i];

delete [] mas;

32. Массивы и их связь с указателями. Адресная арифметика.

Имя массива является указателем на его первый элемент. (&m[0] == m[0])

С помощью указателя на первый эл-т и смещения можно получить доступ к эл-там массива: *(int m[3] = {}; *(m+1) = 1; // m = 010)*

Двумерный массив: (int m[3][3] = {}; int * p = &m[0][0]; *(p+5) = 3);)

// 000

// 003

// 000

33. Передача массивов в качестве параметров ф-ции.

Для одномерных массивов:

void f(int m[], const int size);

void f1(int m, const int size);*

int main() {const int N=10; int mas[N]; f(mas, N); f1(mas, N);}

Для двумерных массивов:

const int N=3, M=2;

void f(int m[][N], int nrow, int ncol);

*void f1(int**m, int nrow, int ncol);*

int main() { int b[M][N]; f(b,M,N); f1(m, M, N);}

34. Концепция памяти. Операции для работы с динамической памятью. Выделение и освобождение памяти под переменные, одномерные массивы.

Область памяти	назначение
Стек (stack)	Область памяти в которой хран. локальные переменные и пар-ты ф-ций
Динам. Память (heap)	Позволяет программисту выд. Память под переменные. Переменные в этой памяти называются динамическими.
Сегмент данных	Для хранения глобальных переменных
Сегмент кода (код программы)	Используется для хранения кода программы. Помещается туда на этапе компиляции и удаляется после завершения программы.

Операции для работы с динам. памятью:

Операция выделения памяти:

new имя_типа; или new имя_типа [размерность];

Возвращает указатель на область выделенной памяти или если объект массив – указатель на его нулевой эл-т или 0, если не может выделить память.

Int *p = newint; int *q = new int(10); //*q=10

int *m = new int[10];

Операция освобождения памяти:

delete выражение;

delete p; delete q; delete [] m;

35. Подпрограммы (синтаксис, виды подпрограмм, контекст, пример).

Подпрограммы в C/C++ представляют собой функции.

Тип имя_ф-ции (список параметров) { объявл. переменных; операторы; return выражение; }

Тип возвращаемого ф-цией значения не может быть массивом и ф-цией.

Если ф-ция не должна ничего возвращать – тип void.

Список параметров опред. величины, которые требуется передать в ф-ю при её вызове.

int func(int x) { return x+1; } int main() { cout « func(5); // 6 }

Параметры ф-ции могут иметь значения по умолчанию

int func(int x, int y=3) { return x+y; } int main() { cout « func(5); // 8 }

Оператор return – осуществляет возврат из ф-ции и передавая выполнение в точку её вызова.

36. Объявление и определение функций. Оператор return.

Смотри билет 35.

37. Способы передачи параметров в функцию

Передача по значению:

В стэк заноятся копии значений параметров. С ними проводятся манипуляции. Доступа к исходным данным у функции нет, она работает с формальными переменными.

Передача по адресу:

В стэк заносятся копии адресов аргументов. Ф-я осущ. доступ к ячейкам памяти по этим адресам => может измен. исх. знач. аргументов.

Void func(int *p) { *p=5; } int main { int x=3; func(&x); // x=5 }

Передача по ссылке:

В ф-ю передается адрес указ. При вызове параметра внутри ф-ции он неявно разыменовывается.

Использование ссылок вместо передачи по значению более эффективно, т.к не требует копирования параметров. Если требуется запретить изм. парам. используется const

void func(int &p) { p=5; } int main { int x=3; func(x); }

38. Способы передачи значения из одной функции в другую.

Использование глобальных переменных, либо передача по ссылке, либо по значению (смотри билет 37)

39. Параметры функции со значениями по умолчанию.

Параметры функции по умолчанию задаются в прототипе функции. Если в функции несколько параметров, то параметры, которые опускаются при вызове должны находиться правее остальных.

int func(int a, int b=1, int c=1) { return a+b+c; } int main { cout « func(3); // 5

cout « func(3, 2, 1); // 6 }

40. Функции с переменным числом параметров.

Переменный список параметров задается в прототипе ф-ции многоточием: *int f(int k…)*

Список параметров совсем пустым быть не может, должен быть прописан хотя бы один явный параметр, адрес которого мы можем получить.

Необходимо предусмотреть способ определения количества параметров и их типов. Для этого есть два способа:

– один из параметров опред. Их число;

– в списке явных параметров задается параметр, указывающий на конец списка параметров;

int sum(int n…) { int s=0, *p=&n; for(int i=0;i<n;++i) s+=*(++p); return s; } int main() { cout « «1+2=» « sum(2, 1, 2); // 3 }

41. Перегрузка функций.

Это определение нескольких ф-ций с одинаковым именем, но различными параметрами. Она нужно для того, чтобы избежать дублирования имён ф-ций, выполняющих сходные действия, но с различной программной логикой.

// ф-ция max для целых чисел

Int max(int num1, int num2) { if (num1 > num2) return num1; return num2; }

// с плавающей запятой

double max(double num1, double num2) { if (num1 > num2) return num1; return num2; }

int main() { cout « max(1,2); // 2 cout « max(2.3,2.1); // 2.3 }

42. Функция main(). Передача параметров в функцию main.

Функция с именем main – начальная точка выполнения для всех программ на языке C/C++

Её можно объявить таким образом:

int main(int argc, char argv[])*

Параметр argc содержит количество параметров, передаваемых в функцию (всегда не меньше 1), параметр argv[] – это массив указателей на строки. Пример исп.:

int main(int argc, char argv[]) { if (argc > 1) cout « argv[1]; else cout « «Not arguments»; }*

43. Указатель на функцию. Передача имен функций в качестве параметров.

Указатель на функцию – переменная, которая содержит адрес некоторой функции. Обращение по этому указателю представляет собой вызов функции.

Объявление указателя: тип (*указатель)(входные параметры). Тип указателя и входные параметры должны быть такие же как у функции.

int sum(int n1, int n2) return n1+n2;

int main() { int (*pf)(int,int) = sum; cout « (*pf)(1,2);//3 }

Или же можно так: *int (*pf)(int,int); pf=∑ (или просто pf=sum;)*

44. Представление строк в языках программирования. Строки в С. Основные алгоритмы обработки строк.

Строка – последовательность символов, заключённых в двойные кавычки.

В С отсутствует спец.строковый тип.

В С/С++ строка представляет собой массив эл-тов типа char, заканчивающийся нулевым символом (нуль-терминатор, нуль символ). Нулевой символ – символ с кодом равным 0, записывается в виде „\0“

Объявление: *char str[10];*

можно при определении инициализировать: *char str[10]= «hello»; // hello\0\0\0\0\0*

Если строка при опред. Инициализируется, её размерность можно опускать: *char str[]= «hello»; // hello\0*

идентично: *char str[]={'H', 'e', 'l' , 'l', 'o', '\0'};*

К любому эл-ту строки, можно обратиться как к эл-ту массива: str[5] = „!“;

Имя массива str – константный указатель на начало строки. (Его значение нельзя менять!)

Указатель на строковую константу: Строка в данном случае – константный массив, размещенный в сегменте данных программы, изменять который нельзя.

Char * st1 = «abcd»; //abcd\0

st1= «def»; // указатель ссылается на новую строку

st1[1]= „w“; // ошибка, т.к пытаемся изменить строковую константу

Ввод строки с клавиатуры: *#include <windows.h>*

*char * str = new char[100];*

SetConsoleCP(1251); SetConsoleOutputCP(1251);

cin » str; // или cin.getline(str, 100);

45. Работа с символами. Основные ф-ции стандартной библиотеки <cctype>.

Различают две группы ф-ций: ф-ции выполняющие классификацию символов и ф-ции выполняющие преобразование символов.

Функции проверки категории символа	
isalnum	Провер. является ли аргумент буквой или цифрой
isalpha	Проверяет является ли аргумент буквой (вернет 1)
iscntrl	// … // управляющим символом
isdigit	// … // цифрой
isgraph	// … // символом, имеющим граф. представление
islower	// … // буквой в нижнем регистре
isprint	// … // символом, который может быть напечатан
ispunct	// … // символом, имеющим граф. Представление, но не являющимся при этом буквой или цифрой
isspace	// … // разделительным символом
isupper	// … // буквой в верхнем регистре
isxdigit	// … // цифрой в 16 СС
Функции изменения регистра	
tolower	Прописную на строчную
toupper	Строчную на прописную

Для каждой из них есть аналог для wchar_t содержащий в названии букву w: iswalpha…

tolower(„A“); // вернет 97 — код символа а

isdigit(„A“); // вернет 0 потому что не цифра

46. Работа со строками. Основные функции стандартной библиотеки <cstring>

Копирование	
strcpy	Копировать одну строку в другую
strncpy	Копировать заданное число символов одной строки в другую
Объединение строк	
strcat	Дописать одну строку к другой
strncat	Дописать заданное число символов из одной строки в другую
Сравнение	
strcmp	Сравнить две строки int strcmp(const char*s1, const char*s2); Возвращ. Значения: <0 — s1 <s2 0 – s1 = s2 >0 — s1 >s2
strncmp	Сравнить заданное число первых символов 2 строк
Поиск вхождения символа	
strchr	Найти первое вхождение символа в строку char*strchr(char*s, int ch); Возвращает указатель на 1 вхождение символа ch в строку s, если его нет – возвращает NULL
strcpsn	Найти первое вхождение символа из заданного мн-ва в строку
strpbrk	Осуществить послед. Поиск всех вхождений символов заданного мн-ва в строке
strrchr	Найти последнее вхождение символа в строку
strspn	Определить сколько символов заданного мн-ва встречается с начала строки
strstr	Найти первое вхождение подстроки в строку
strtok	Разбить строку по разделителю

другие	
strlen	Вычислить длину строки

cout « strlen(«string»); // 6

cout « strcmp(«string», «string»); // 0 — одинаковые строки

char str1[]= «string1»;

char str2[] = «string2»;

strcpy(str1, str2);//копируем str2 в str1

cout « str1 « endl « str2; // string2\nstring2

47. Файловый ввод-вывод. Стандартная библиотека ввода-вывода (cstdio). Типовые алгоритмы обработки файлов.

Работа с потоком начинается с его открытия.

FILE fopen(const char*filename, const char*mode);*

fopen() открывает поток и связывает с этим потоком определенный файл.

При успешном открытии возвращает указатель на структуру типа FILE, если ошибка – пустой указатель (NULL).

Режимы открытия файла: «r» – чтение, «w» – открыть файл для записи (если файл сущ. Он стирается), «a» – для добавления информации в конец; «r+» – для чтения и записи (файл должен существовать), «w+» – файл для чтения и записи (если существует, он стирается), «a+» – для чтения и добавления информации в его конец

Режим открытия может содержать символы t (текстовый режим) или b (двоичный). (например «rb», «a+b», «ab+»)

char filename= «text.txt»;*

*FILE*f=fopen(filename, «w+»);*

if(!f) cout « «Ошибка при открытии файла»;

Закрытие потока: *int fclose(FILE* f);*

Возвращает 0, если операция закрытия файла прошла успешно, EOF – в случае ошибки. Чтобы точно узнать в чем ошибка можно использовать ferror();

Обработка ошибок: *int feof (FILE*f)* возвращает не равное нулю значение, если достигнут конец файла, в противном случае 0;

Позиционирование по файлу: int *fseek(FILE*f, long offset, int origin);* функция перемещает указатель позиции в потоке. Устанавливает внутренний указатель положения в файле на новую позицию, путем добавления смещения offset к исх. Положению origin.

Origin может быть константой: SEEK_SET – начало файла, SEEK_CUR – текущее положение файла, SEEK_ED – конец файла;

*int fgetpos(FILE * filestream, fpos_t * position);* – Получить значение текущего положения указателя в файле.

*int fsetpos(FILE * filestream, const fpos_t * pos);* – Функция перемещает внутренний указатель положения в файле, связанный с потоком, на новую позицию.

Ввод/вывод:

*Int fgetc(FILE*f);* – возвращает очередной символ (в форме int) из потока f. Если символ не может быть прочитан (конец файла или ошибка) – EOF

*int fputc(int ch, FILE*f);* – если успешно, вернет записанный символ, иначе – EOF;

*char *fgets(char *s, int n, FILE *f);* – читает из потока f не более n-1 символов и помещает их в массив, адресуемый указателем s; Символы читаются пока не будет прочитан символ новой строки или конца файла или пока не будет достигнут заданный предел. Возвращает NULL при обнаружении ошибки или конца файла, в противном случае – указатель на строку s.

*int fputs(const char *s, FILE *f);* – Функция записывает строку символов s в поток f. Символ конца строки в файл не записывается. При ошибке возвращает значение EOF, иначе — неотрицательное число.

*size_t fwrite(const void * buf, size_t size, size_t n, FILE * f);*

Функция записывает n элементов длиной size байт из буфера, заданного указателем buf, в поток f. Возвращает число записанных элементов.

*size_t fread(void * buf, size_t size, size_t count, FILE *f)*

Функция считывает count элементов size байтов в область, заданную указателем buf, из потока f.

*int fprintf(FILE *f, const char * управляющая_строка, ...);*

*int fscanf(FILE *f, const char * управляющая_строка, ...);*

Работа данных функций полностью идентична printf() и scanf() соответственно, за исключением того, что они перенаправляют поток ввода/вывода в файл